published as: Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, Christoph Meinel: Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses, In Proceedings of 4th Annual IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 10-12 December 2015, Zhuhai
2015 ANNUAL IEEE INTERNATIONAL CONFERENCE ON TEACHING, ASSESSMENT, AND LEARNING FOR ENGINEERING

Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses

Thomas Staubitz[1], Hauke Klement[2], Jan Renz[1], Ralf Teusner[1], Christoph Meinel[1]
Internet Technologies and Systems
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
[1]Firstname.Lastname@hpi.de, [2]Firstname.Lastname@student.hpi.de

***Abstract*—In recent years, Massive Open Online Courses (MOOCs) have become a phenomenon presenting the prospect of free high class education to everybody. They bear a tremendous potential for teaching programming to a large and diverse audience. The typical MOOC components, such as video lectures, reading material, and easily assessable quizzes, however, are not sufficient for proper programming education. To learn programming, participants need an option to work on practical programming exercises and to solve actual programming tasks. It is crucial that the participants receive proper feedback on their work in a timely manner. Without a tool for automated assessment of programming assignments, the teaching teams would be restricted to offer optional ungraded exercises only. The paper at hand sketches scenarios how practical programming exercises could be provided and examines the landscape of potentially helpful tools in this context. Automated assessment has a long record in the history of computer science education. We give an overview of existing tools in this field and also explore the question what can and/or should be assessed.**

Keywords—MOOC, Massive Open Online Courses, Programming, Assessment, Automated Assessment

I. INTRODUCTION AND MOTIVATION

Massive Open Online Courses (MOOCs) provide a scalable and socially interactive learning experience. High-quality courses, covering various subjects, are made freely available to anyone connected to the Internet. MOOCs [1] have a tremendous potential to introduce a large and diverse audience to the basics of programming. Introductory courses in CS and engineering, which are already offered by the majority of MOOC providers, are regarded to be an adequate means to attract students into the subject [43]. Other courses aim at introducing teachers to new topics that can improve the appeal of their teachings [27]. In our own experience introductory programming courses attract large amounts of participants from all ages and all backgrounds. At the time of writing, we have offered two programming courses aiming at beginners. One in Python, another in Java. The Python course was marketed as a course for school children but in the end attracted participants from age 11 to 82. Within the scope of an initiative to create new jobs in the digital sector, the European Commission published a study [11] investigating the demand and supply of MOOCs related to web skills. Results of an associated survey show that IT professionals consider MOOCs the best way to learn such abilities. The responses to the survey also indicate that learners are less interested in theoretical content but value practical experience. According to the study, neither the standard formulas of academic courses nor the prevalent MOOC format are optimal for teaching web-related skills. Instead, survey participants noted the importance of learning-by-doing practices. Berges et al. [8] report that in courses on object oriented programming (OOP) students show radical differences in the way several groups of concepts are grasped. Particularly, they differentiate between the students understanding of these concepts and their ability to apply them practically. Learning to program does not only involve acquiring complex knowledge but also related practical skills [35]. Therefore, gaining programming expertise requires rigorous practice [43].

Programming assignments can help students to become familiar with programming languages and tools, and to understand how the principles of software design and development can be applied in practice [12]. On-campus programming courses usually make use of practical assignments that build up on theoretical content presented in lectures. These assignments are regarded to be an indispensable part of the educational framework [32] and are used for assessment by the majority of

[1]In general, the literature differentiates between cMOOCs and xMOOCs. We use the term MOOC in short for xMOOC. For a detailed distinction between cMOOC and xMOOC please see e.g. https://eleed.campussource.de/archive/10/4074

CS academics [39]. According to Feldman and Zelenski [16] the major part of students' learning outcomes in a beginners' programming course originates from completing programming projects. The most important deficits of novice programmers relate to designing problem solutions and express them as actual programs. Frequent practical programming exercises are a common way for addressing these issues [35]. A complete solution to a programming task is considered to be an important step in building the confidence of student programmers [12]. These are just a few examples that indicate the importance of practical exercises in different areas of programming education.

Traditionally, MOOCs are composed of video lectures, reading material, and assessment tools that are limited to a set of automatically gradable assignment types, such as quizzes. However, these means are not sufficient for teaching programming, which requires practice, feedback, and code assessment. According to Neuhaus et al. [32], the current generation of MOOC platforms is well suited for presenting teaching material, but it provides only inadequate possibilities for hands-on experiments. Supported assignments are essentially non- interactive and do not allow a step-by-step development of solutions. However, in order to enable a more holistic learning process, MOOCs need to integrate activities that allow active experimentation and that relate to concrete experience [20]. Willems et al. [48] state that the implementation of systems that allow the assessment of practical exercises can be a great challenge for course creators and platform designers. Nevertheless, the authors see the ability to offer classes with a high share of practical tasks and assignments as a key feature of MOOC platforms, which will have a crucial impact on a platform's competitive position.

In order to provide an attractive and supporting platform for teaching programming to the masses, MOOCs have to fit the requirements of programming education. While MOOCs can deliver course contents to tens of thousands of students, providing appropriate tools for practice and offering assessable practical programming assignments usually exceeds their built-in capabilities.

Our general research examines the question how to extend MOOC platforms in order to provide richer technical support for hands-on learning and collaboration, which are key aspects in terms of addressing the criticism that MOOCs, too often, are based on questionable teaching strategies, such as behaviorism [4], [37].

In this context, the paper at hand defines a starting point regarding the question how MOOCs can integrate practical programming assignments in a manner that meets the demands of novice programmers, provides an efficient and easy-to-use solution for the teaching teams, and satisfies the inherent scalability requirements of large-scale e-learning environments. We sketch the landscape of existing tools that are potentially helpful in addressing the tasks of providing practical programming exercises and automated assessment. Furthermore, we explore the literature on the long record of automated assessment solutions in the history of computer science education

II. COMPONENT LANDSCAPE

We identified basically four fundamentally different scenarios to provide practical programming tasks with automated assessment in MOOCs.

- Scenario 1: The user installs some sort of development software locally. The platform only provides the description of the exercise and, if necessary, required additional materials. The user in return uploads her solution to the platform for automated assessment.
- Scenario 2: Instead of using locally installed development software, third party online coding tools are employed. Apart from that, scenario 2 is identical to scenario 1.
- Scenario 3: The platform itself features a development environment. Exercises are provided and assessed in this environment. Code execution and assessment is handled on the server side.
- Scenario 4: Identical to scenario 3 except for client-side code execution.

Each of these scenarios has its benefits and drawbacks. The main benefit of scenario one, two, and four is that there are hardly any scalability problems as execution is handled on the client-side and assessment can be handled asynchronously. For courses addressing beginners, scenario one could benefit from employing specialized coding tools with an educational background, such as BlueJ[2] or Greenfoot[3]. For courses addressing more advanced target groups, scenario one would enable the participants to work with their preferred and familiar tools. The main drawback of scenario one, particularly for beginner's level courses, is the heterogeneity of operating systems, code editors, IDEs, compilers, interpreters, additional libraries that need to be installed, which is predestined to cause an increased amount of support requests that can hardly be handled. This effect can be diminished by providing a virtual machine that already contains all required prerequisites. Scenario two needs to take in consideration that the third party tool should be prepared for sudden increases of user numbers, when promoted in a MOOC. Another major drawback of the first and the second scenario is that only the final results of the participant's development process will be stored at the server to be analyzed in post course research projects. A major benefit of scenarios three and four is that they allow collecting partial solutions and reproducing the iterations of a learner's development cycle. This can provide valuable insights into students' problem-solving strategies.

The following discusses the four scenarios in more detail and introduces some tools that might be helpful or inspiring for one or the other of these scenarios.

Web-based development tools as suggested in scenario two, three, and four provide homogeneous, installation-free development environments. By eliminating the need for setup and configuration, they lower students' barriers to start programming [45]. Since participants of a MOOC already have

[2] http://www.bluej.org/

[3] http://www.greenfoot.org/door

access to a web browser, web-based development tools are virtually predestined in this context [51]. Furthermore, the web-based nature of MOOC platforms enables a tight integration of web-based special-purpose tools. Web-based development environments can either be provided by bringing dedicated tools into operation (scenario three and four) or by leveraging third-party tools that are already existent (scenario two).

Dedicated development tools are supplied as tightly integrated parts of the MOOC platform. Tight integration can also be achieved if the development tool is only loosely coupled with the platform, e.g. by employing the Learning Tools Interoperability (LTI)[4] standard for data exchange between MOOC platform or Learning Management System (LMS)[5] [6]. Embeddable JavaScript code editors, such as Ace[7] and CodeMirror[8] could serve as the basis for such a dedicated development tool. They offer rich code editing capabilities that are comparable to those provided by native desktop editors.

Dedicated development tools can be distinguished based on their approach for the execution of learners' code. Student-written code is either executed in the client's web browser or transmitted to the server for remote execution.

Executing a learner's code on her own machine is a resource-efficient approach since no server-side resources are claimed for code execution. Furthermore, there is no need for security considerations in terms of dealing with potentially untrustworthy code. Moreover, since no client-server round trips are involved, client-side code execution promotes interactivity and avoids potential delays during high-demand periods before assignment deadlines [28]. Using the learner's web browser as execution platform is particularly suitable for teaching client-side web technologies, such as Hypertext Markup Language (HTML), JavaScript, and Cascading Style Sheets (CSS), since interpreters for these languages are built into browsers. The major drawback of client-side code execution is its limitation to browser-supported programming languages and APIs as well as special JavaScript-based derivatives of non-native languages, such as ClojureScript[9], Opal[10], and Skulpt[11], which are in-browser implementations of Clojure[12], Ruby[13], and Python[14].

Compared to its client-side equivalent, server-side code execution offers much more flexibility since the set of executable programming languages is virtually unlimited. Moreover, code evaluation for both exploration and assessment is performed in one place and can use the same procedure. Additionally, sending partial solutions for execution to the server allows reproducing the iterations of a learner's development cycle and can provide valuable insights into students' problem-solving strategies. The advantages of server-side code execution come at the cost of increased computational load and feedback latency. Furthermore, careful security considerations are necessary.

Web-based development tools can also be realized without the need for self-hosted solutions. Instead of providing dedicated development environments and allocating platform resources, programming MOOCs can leverage third-party services for several or even all aspects of the development process [17], [39]. Software as a Service (SaaS) and Platform as a service (PaaS) providers typically offer free plans for starters, which fit the needs of MOOC participants and can provide the tools that are needed for practical programming assignments. For instance, novice programmers' demands could be covered based on third-party services by leveraging Cloud9[15] as a web-based IDE, GitHub[16] for code hosting and issue tracking, Heroku as execution platform, and Travis CI[17] for continuous testing.

Not only does this approach save the resources of the e-learning platform, but it also enables learners to gain practice in working with tools and services that are used by professionals. According to Fox and Patterson [17], deploying their projects in the same scalable environment as used by professional developers supplies learners with valuable experience. Moreover, the approach can provide students a feeling of accomplishment when shipping working code that can be used by people other than their instructors. Relying on freely available online services involves the drawbacks that learners are required to register with third-party companies, that individual tools are spread over different platforms, and that MOOCs following this approach are highly dependent on the availability and reliability of external parties.

Third party educational web-based tools, such as Codewars[18] and CodingBat[19] can assist the teaching teams as they already supply collections of practical programming problems to be solved in the web browser. These tools do not provide a course framework, but they can support novices on their way to mastery by offering an engaging opportunity to practice.

Educational programming games are designed to maximize the appeal of learning to program. Learners' motivation is raised by using inciting game elements, such as increasingly challenging levels, scores, and leaderboards. CodeHunt[20] [41] is a web-based coding game, aimed at teaching programming at scale. It challenges students to complete skeletal methods, given in either Java or C#, so that they satisfy a hidden

[4] http://www.imsglobal.org/toolsinteroperability2.cfm
[5] LTI is supported by a wide variety of LMS.
[6] http://www.imsglobal.org/cc/statuschart.cfm
[7] http://ace.c9.io/
[8] http://codemirror.net/
[9] http://clojurescript.net/
[10] http://opalrb.org/
[11] http://www.skulpt.org/
[12] http://clojure.org/
[13] https://www.ruby-lang.org/
[14] https://www.python.org/
[15] https://c9.io/
[16] https://github.com/
[17] https://travis-ci.org/
[18] http://www.codewars.com/
[19] http://codingbat.com/
[20] https://www.codehunt.com/

specification, which is only given by input/output (I/O) pairs. Similarly, Xiao and Miller [50] describe a multi-player online programming game that is aimed at teaching novice CS students' best practices for collaborative programming in large software projects.

Online development tools such as CodePen[21], jsFiddle[22], and repl.it[23] have no primary educational objective, they provide developers with in-browser programming environments for impromptu development and execution of short programs. Such platforms' use cases include trying out libraries, constructing minimal programs for troubleshooting, and sharing code snippets. While CodePen and jsFiddle focus on the combination of Javascript, CSS, and HTML, repl.it supports a little wider variety of languages.

Full-featured web-based integrated development environments (IDEs) are mentioned in research [1], [19], [47], [49] and are available as open-source software or hosted solutions, for example by Cloud9[24], Codio[25], and Nitrous.IO[26]. Web- based IDEs usually make use of traditional desktop user interface (UI) patterns, such as menu bars, file trees, content tabs, context menus, and drag-and-drop operations. Besides sophisticated code editing capabilities, such applications' features may include customizability, project management, version management, and full Linux environments for building and executing applications. Since computationally intensive tasks are performed on a remote server, low-end PCs and mobile devices can be used as development machines.

Web-based IDEs often facilitate the deployment of applications to infrastructures supplied by PaaS providers, such as Google App Engine[27], Heroku[28], and Microsoft Azure[29]. Therefore, anybody with modest software development skills is able to deploy applications to the Cloud with small effort and low budget [1].

Another feature that is predestined for web-based IDEs is collaborative editing, as known from Etherpad[30] and Google Docs[31]. Multiple developers who are working at the same time are provided with a consistent view of a project since they receive real-time updates of their collaborators' changes. Collaborative coding facilitates side-by-side pair programming, benefits communication and team knowledge sharing, and may increase productivity and software quality [19].

[21] http://codepen.io/
[22] https://jsfiddle.net/
[23] http://repl.it/
[24] https://c9.io/
[25] https://codio.com/
[26] https://www.nitrous.io/

[27] https://appengine.google.com/
[28] https://www.heroku.com/
[29] http://azure.microsoft.com/
[30] http://etherpad.org/
[31] https://docs.google.com

III. AUTOMATED ASSESSMENT

Design Challenges—High-quality assignments are seen as a vital part of a successful course [16]. While manual assessment allows compensating for poor assignment design, the use of automated assessment techniques increases the need for carefully designed assignments [33]. The creation of automatically assessable programming assignments is considered a challenging task that requires special attention [2]. Whereas automated assessment saves instructors' time by outsourcing formerly manually performed grading activities, a considerable amount of the gained time should be allocated for designing and implementing resources for automated assessment. While efforts may only be shifted from grading activities to design activities for small class sizes, the trade-off increasingly shows its strengths with rising student numbers. Whenever assessment is performed without human intervention, the assignment specification should be provided as unambiguous as possible. Ambiguous specifications permit different interpretations, which can lead to technically valid student solutions being rejected by an automatic grader. Within programming, interpretation is key to success, which is why assignment instructions must guide interpretation precisely for successful automated assessment [12]. In contrast, careless formulation of assessment criteria can result in improper assessment [33]. Therefore, ambiguity must be minimized in order to increase fairness and quality of assessment [34]. Cerioli and Cinelli [9] even regard an extremely precise problem specification, which allows a completely predictable behavior of implementations, as a prerequisite for automated grading based on functional correctness. However, a reasonable balance between the risk of misinterpretation and excessive detail has to be found because wordier specifications, which point out every detail, can result in trivial assignments lacking any demand to reason about the problem [34]. Besides addressing the problem of ambiguity, the definition of pedagogically sound test cases is a time-consuming activity [9] that requires both expertise and experience [43]. Pieterse [33] names test data "the Achilles' heel of any system that applies automated assessment of programming assignments". In order to enable accurate assessment and prevent incorrect solutions from passing the evaluation, tests must be designed well. Otherwise, learners might submit deficient solutions but remain unaware of their incorrectness.

Approaches—for performing automated assessment of programming assignments can be categorized into dynamic approaches, which require execution of the program under test, and static approaches, which do not. While most approaches focus on evaluating the functional completeness and correctness of a program, others aim at evaluating aspects of quality and style.

I/O-based Assessment–refers to assessing a program solely by using a standard I/O interface. The program under test is supplied with predefined values and is verified to produce expected output values. The advantage of this approach is its versatility. I/O-based assessment can be applied to any program using an I/O interface and to any programming language that can be executed on the same test environment [24]. Moreover, test cases may be reused across multiple languages since a universal interface is sufficient for their

execution. A shortcoming of the approach is that it may fail to give an appropriate mark if a student program's output does not exactly match the expected format [33]. Therefore, I/O- based assessment techniques are not usable if strict format requirements are not feasible or if freedom in formatting should be allowed. However, implementing I/O handling that is robust to irrelevant output differences, such as whitespace and orthographic mistakes, is a challenge [12]. Due to lacking insights into the inner mechanics of a code submission, I/O-based assessment is limited to testing side effects that are exposed in the form of program output. For the same reason, I/O-based assessment is not qualified for providing the learner with feedback regarding why her submission deviates from the specification.

Assessment Using Industrial Testing Tools–In-depth feedback can be provided by utilizing industrial-strength testing tools and frameworks. Such tools are widely used, are actively developed, and can supply deeper insights into the program under test. Since testing is an established practice in industry, myriads of testing frameworks exist for virtually every programming language and application domain. Ihantola et al. [24] name three classes of industrial testing tools that are used by automated assessment systems: xUnit-based frameworks, acceptance testing frameworks, and web testing frameworks. xUnit is a collective term for numerous testing frameworks that derive their design from SUnit [5], an influential testing framework for Smalltalk, which is considered "the mother of all unit testing frameworks" [13]. Widespread xUnit derivatives include CUnit[32] for C, HUnit[33] for Haskell[34], and JUnit[35] for Java. These language-specific testing frameworks enable assessment techniques that can evaluate the functionality of entities smaller than a complete program, such as single classes, methods, and even statements [2].

Acceptance testing–refers to an industrial testing technique that is based on customers specifying test scenarios that have to be passed so that user stories are considered to be correctly implemented. This testing approach helps customers and developers to foster a common understanding of how software under development should work once it is finished. Acceptance testing frameworks, such as Cucumber[36], FitNesse[37] , and Lettuce[38], usually rely on easily understandable plain-text domain-specific languages (DSLs), similar to natural language. This allows non-technical stakeholders to contribute their domain knowledge by providing scenarios that specify navigation through the application, inputs to the application, and expected outputs [17]. Scenarios are turned into executable tests whose successful execution is to be achieved. When used for student assessment, acceptance testing offers the advantage that a single specification can serve as both assessment basis and exercise instructions since it is given in easily understandable form and expected to be complete. Web testing frameworks, such as Selenium[39] and Watir[40], are useful tools for assessing web application exercises. Instead of accessing low-level application programming interfaces (APIs), web testing tools test web applications using their public web interfaces. This can either be done by controlling a real web browser in an automated fashion or by simulating a web browser by means of Hypertext Transfer Protocol (HTTP) requests.

Assessment of Testing Skills–Modern software development processes, such as Scrum [38] and Extreme Programming (XP) [6], promote test-first practices, which help to discover design flaws as early as possible in the development cycle and underline the value of regression tests for continuous delivery. When novice programmers are assessed using traditional automated approaches, they are neither encouraged nor rewarded for performing testing themselves since an automated grader verifies their programs' correctness anyhow. As a result, learners might not reflect upon the behavior of their code, but they might solely focus on providing a solution that satisfies the automated approach [14]. However, efficient automatic testing approaches should not invite students to get careless. Instead, students should learn to design and test their programs thoroughly before submitting them [2]. In this sense, Edwards [14] argues that students need to acquire software testing skills. He suggests exposing students to test-driven development (TDD), so that they perform more testing and eventually appreciate its value for the development process. Moreover, a testing-oriented assessment approach empowers students with the responsibility of demonstrating their own programs' correctness and validity. As a result, the learning experience is enhanced and learners produce higher-quality code. According to Pieterse [33], the application of test-based assessment combined with training in software testing can provide a learning experience where students learn to favor robust and precise solutions over improvised ones. The term meta testing refers to a test-based assessment approach that evaluates students' software testing skills. Instead of providing learners with prepared tests, be it explicitly as visible part of an exercise or implicitly as the basis for program evaluation, this approach demands learners to write tests themselves. They are required to submit working program code along with proper tests. Grading can be based on judging the extent to which the student-written code fulfills the accompanying tests, the tests' level of quality, and the fraction of code covered by tests. Additionally, the teacher might incorporate her own tests into the assessment in order to validate that the student's submission indeed satisfies the exercise specification.

Assessment of GUI Applications–Even though focusing solely on command-line interface (CLI) applications may be perfectly sufficient for conveying programming skills, such an educational approach may be seen as uninspiring by learners to whom graphical applications are familiar and much more attractive than CLI-based ones [12]. Instead, students are interested in learning how to build programs with GUIs [15]. Likewise, applications that produce animations or perform 3D

[32] http://cunit.sourceforge.net/
[33] http://hunit.sourceforge.net/
[34] https://www.haskell.org/
[35] http://junit.org/
[36] http://cukes.info/
[37] http://www.fitnesse.org/
[38] http://lettuce.it/
[39] http://www.seleniumhq.org/
[40] http://watir.com/

rendering are usually appealing to learners. However, GUI applications are difficult to assess since I/O redirection, as used for the assessment of CLI applications, is not applicable. Developing software tests for programs involving significant GUIs is ranked beyond the typical abilities of students and educators [40]. A response to this problem are educational GUI libraries, such as presented by English [15] and Thornton et al. [40]. These libraries are designed for novice programmers and provide built-in means for automated testing and assessment. Moreover, the latter framework is explicitly aimed at allowing students to write tests themselves. Therefore, it can enable automated assessment for GUI applications that follows a TDD-based assessment approach.

Assessment of Style–Besides functional completeness and correctness, there are further aspects that are crucial to the quality of software, such as its complexity, extensibility, and maintainability. Writing code in good style is important because program code is read much more often than it is written [36]. Since software projects are usually carried out in groups, developers need to follow established coding conventions that facilitate a common understanding among them and guarantee a certain degree of quality. In general, good coding style promotes readability, absence of errors, security, extensibility, and modularity [36]. However, novice programmers are reported to commonly perceive programming style as less significant [3] and to have little appreciation for best practices, which are required for successful long-term multi-person programming projects [50]. Therefore, programming style is an important issue to teach beginning programmers. It is often neglected in education, though [3]. Automated techniques can help to involve programming style into the assessment process. In contrast to functionality, which is usually assessed by executing a program submission, properties of style are typically collected using static evaluation approaches. A common practice for judging a student program's quality is detecting so-called code smells, such as unused variables, redundant logical expressions, and implicit constants [42]. Furthermore, automated style evaluation can examine programs' adherence to given coding guidelines in terms of indentation size, mandatory source code documentation, and more [3]. High-complexity program submissions can be detected by employing software metrics, such as Halstead's complexity measures [21] and McCabe's cyclomatic complexity [31], and by comparing their structure to that of a model solution [42].

Peer Assessment–There are program characteristics that are hard to assess automatically, however; for instance, quality of comments, meaningfulness of variable names, and adherence to good practices, such as the Single Responsibility Principle [30]. Evaluating such subtle or complex software properties requires the trained eye of a human assessor. Peer assessment could be a possible alternative in such cases. It is beyond the scope of this paper, however.

Automated Assessment Tools—Automated techniques have been used for the assessment of programming assignments almost as long as programming has been taught [33]. Automated assessment approaches are used to keep teachers' workload within reasonable limits despite growing student numbers [45]. This way, the time required for assessment activities can be cut down without reducing quantity and quality of practical exercises. Furthermore, the amount of time that instructors can spend on mentoring and supporting students is increased [44]. Automated program evaluation can also be beneficial to students. While human graders and especially teams of multiple graders usually judge subjectively and inconsistently, automated assessment can provide objective and consistent evaluation [2]. Furthermore, students are provided with immediate feedback, which is an important benefit in programming education. Receiving instant feedback is particularly useful for novice programmers since misconceptions are uncovered as early as possible [46]. The concept of providing feedback at any time and any place applies notably well to virtual courses [29], such as MOOCs, and provides learners a unique advantage [10]. Since assessment resources are virtually unrestricted, automated grading allows students to increase mastery by iteratively improving and resubmitting their homework [18].

Systems that automatically assess students' programming assignments have been designed and used for over fifty years. Systematic overviews of assessment systems' approaches and capabilities have been published by Ala-Mutka [2], Douce et al. [12], and Ihantola et al. [24]. Douce et al. present a historical overview of automated assessment systems that focuses on systems that are based on executing tests in an automated fashion. The authors classify these systems into three generations. The first generation covers the initial attempts to automate the assessment of programming assignments. In general, first-generation systems were specifically tailored solutions that required modifications to compilers and operating systems (OSs), demanded a great deal of expertise, and were limited to the usage in their particular setting. The very first system has been described by Hollingsworth [23]. Its purpose was to evaluate programs written in assembly language, which had to be handed in on punched cards. The system was not only useful for saving teacher resources but also for allocating computing resources, which were severely limited at that time. The second generation of automated assessment systems is characterized by the adoption of automated tools and utilities, provided by increasingly advanced OSs and tool sets. The systems could be operated by instructors and students using a CLI or a graphical user interface (GUI). Second-generation systems introduced more sophisticated assessment strategies involving multiple assessable properties, such as correctness, efficiency, and style. Several systems also include management capabilities for courses and assignments. Well-known representatives of second-generation systems are ASSYST [25], BOSS [26], and Ceilidh [7]. Third-generation automated assessment systems took advantage of advancing web technologies. They comprise features such as web interfaces, increasingly sophisticated testing approaches, interactive feedback, richer content management features, and plagiarism detection. Systems of the third generation include instances of second-generation systems that continued to develop, such as BOSS, and successors of former systems, such as CourseMarker [22], which evolved from Ceilidh. While Douce et al. assign state-of-the-art automated assessment systems to the third generation, their work cannot cover trends that emerged after 2005, for instance the growing demand for practical

assignments in e-learning. In that respect, Ihantola et al. [24] report an increasing interest in extending LMSs with automated assessment capabilities in order to fit the special needs of CS education better. For the same reason, programming MOOCs should be provided with modern capabilities for automatic code assessment.

IV. FUTURE WORK

In the meantime, we already made some progress in the development of an automated assessment tool. We will describe the architecture decisions that allow for a scalable solution, our first experiences using the tool with about 10.000 patient users and the resulting learnings in future papers. Additionally, to that, we will also analyze the submissions we got from the first Java course conducted with that tool with regard to common errors. In courses yet to come, we will also offer enhanced direct interaction between course participants concerning their code, by allowing them to request comments on tricky parts and offering a synchronized video chat. The insights gained there are expected to give hints to improve the assignment descriptions, lecture videos and didactical approaches in general.

V. CONCLUSION

Practical programming tasks are essential for programming courses. Particularly, in the context of MOOCs automated assessment of these tasks is a must, as due to the high enrollment numbers manual assessment is not feasible for teaching teams. Peer Assessment might serve as an alternative, however, e.g. in contexts where automated assessment would require too much effort in preparation. Automated assessment solutions are well established and have a long history, which dates back to the early days of computer science education. Several approaches or scenarios to tackle this task have been identified. Comparing the benefits and drawbacks of the approaches that have been introduced in Section II, we conclude that we need a flexible solution that is able to handle things differently depending on a course's main target group. Beginners benefit more from a browser based environment relieving them from the agony of installation hassles. More advanced users, on the other hand, will prefer to stick with the familiar tools that they already have installed. In terms of server load a "code local–assess remote" approach has the advantage that it not necessarily would require a real-time handling of the assessment. The solutions of the users could as well be queued. Particularly in more advanced contexts, a certain relay in the feedback is not desirable but tolerable.

The landscape of existing programming languages is wide spread and still growing. Providing a new programming environment for each course is not desirable. We, therefore, suggest to develop a versatile tool that is able to automatically assess a variety of programming languages and can deal with local and remote coding scenarios.

REFERENCES

[1] T. Aho, A. Ashraf, M. Englund, J. Katajamäki, J. Koskinen, J. Lautamäki, A. Nieminen, I. Porres, and I. Turunen. Designing IDE as a Service. *Communications of Cloud Software*, 1(1), 2011.

[2] K. Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, 2005.

[3] K. Ala-Mutka, T. Uimonen, H.-M. Järvinen, and L. Knight. Supporting Students in C++ Programming Courses with Automatic Program Style Assessment. *Journal of Information Technology Education*, 3, 2004.

[4] T. Bates. What's right and what's wrong about Coursera-style MOOCs, 2012. [Online; accessed 18-April-2015].

[5] K. Beck. Simple Smalltalk Testing: With Patterns. *The Smalltalk Report*, 4(2):16–18, 1994.

[6] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[7] S. Benford, E. K. Burke, E. Foxley, and C. A. Higgins. The Ceilidh System for the Automatic Grading of Students on Programming Courses. In *Proceedings of the 33rd Annual on Southeast Regional Conference*, ACM-SE 33, pages 176–182, New York, NY, USA, 1995. ACM.

[8] M. Berges, A. Mühling, and P. Hubwieser. The gap between knowledge and ability. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 126–134, New York, NY, USA, 2012. ACM.

[9] M. Cerioli and P. Cinelli. GRASP: Grading and Rating ASsistant Professor. In *Proceedings of the ACM-IFIP IEEIII 2008 Informatics Education Europe III Conference. Venice, Italy*. Citeseer, 2008.

[10] A. Chauhan. Massive Open Online Courses (MOOCS): Emerging Trends in Assessment and Accreditation. *Digital Education Review*, 25:7–17, 2014.

[11] European Commission. Support Services to Foster Web Talent in Europe by Encouraging the Use of MOOCs Focused on Web Talent, 2014.

[12] C. Douce, D. Livingstone, and J. Orwell. Automatic Test-Based Assessment of Programming: A Review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

[13] S. Ducasse. SUnit Explained. Technical report, University of Berne, Institute of Computer Science, 2003.

[14] S. H. Edwards. Improving Student Performance by Evaluating How Well Students Test Their Own Programs, 2003.

[15] J. English. Automated Assessment of GUI Programs using JEWL. *ACM SIGCSE Bulletin*, 36(3):137–141, 2004.

[16] T. J. Feldman and J. D. Zelenski. The Quest for Excellence in Designing CS1/CS2 Assignments. *ACM SIGCSE Bulletin*, 28(1):319–323, 1996.

[17] A. Fox and D. Patterson. Crossing the Software Education Chasm. *Communications of the ACM*, 55(5):44–49, 2012.

[18] A. Fox, D. Patterson, R. Ilson, S. Joseph, K. Walcott-Justice, and R. Williams. Software Engineering Curriculum Technology Transfer: Lessons learned from MOOCs and SPOCs. Technical report, EECS Department, University of California, Berkeley, 2014.

[19] M. Goldman, G. Little, and R. C. Miller. Real-Time Collaborative Coding in a Web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 155–164. ACM, 2011.

[20] F. Grünewald, C. Meinel, M. Totschnig, and C. Willems. Designing MOOCs for the Support of Multiple Learning Styles. In *Scaling up Learning for Sustained Impact*, pages 371–382. Springer, 2013.

[21] M. Halstead. *Elements of Software Science*. Elsevier Science Inc., 1977.

[22] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies*, 8(3):287–304, 2003.

[23] J. Hollingsworth. Automatic Graders for Programming Classes. *Communications of the ACM*, 3(10):528–529, 1960.

[24] P.Ihantola, T.Ahoniemi, V.Karavirta, and O.Seppälä. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM, 2010.

[25] D. Jackson and M. Usher. Grading Student Programs using ASSYST. *ACM SIGCSE Bulletin*, 29(1):335–339, 1997.

[26] M. Joy, N. Griffiths, and R. Boyatt. The BOSS Online Submission and Assessment System. *Journal on Educational Resources in Computing (JERIC)*, 5(3):2, 2005.

[27] J.S.Kay and T.McKlin. The Challenges of Using a MOOC to Introduce "Absolute Beginners" to Programming on Specialized Hardware. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 211–212. ACM, 2014.

[28] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen. A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin*, 37(3):14–18, 2005.

[29] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses. *ACM SIGCSE Bulletin*, 34(3):55–59, 2002.

[30] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, 2003.

[31] T. J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[32] C. Neuhaus, F. Feinbube, and A. Polze. A Platform for Interactive Software Experiments in Massive Open Online Courses. *Journal of Integrated Design and Process Science*, 18(1):69–87, 2014.

[33] V. Pieterse. Automated Assessment of Programming Assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, pages 45–56, 2013.

[34] J. Renz, T. Staubitz, C. Willems, H. Klement, and C. Meinel. Handling Re-grading of Automatically Graded Assignments in MOOCs. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 408–415. IEEE, 2014.

[35] A. Robins, J. Rountree, and N. Rountree. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2):137–172, 2003.

[36] S. Rogers, S. Tang, and J. Canny. ACCE: Automatic Coding Composition Evaluator. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 191–192. ACM, 2014.

[37] R. Schulmeister. The position of xmoocs in educational systems. *eleed*, 10(1), 2014.

[38] K. Schwaber. Scrum Development Process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[39] T. Staubitz, J. Renz, C. Willems, J. Jasper, and C. Meinel. Lightweight Ad Hoc Assessment of Practical Programming Skills at Scale. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 475–483. IEEE, 2014.

[40] M. Thornton, S. H. Edwards, R. P. Tan, and M. A. Pérez-Quiñones. Supporting Student-Written Tests of GUI Programs. *ACM SIGCSE Bulletin*, 40(1):537–541, 2008.

[41] N. Tillmann, J. de Halleux, T. Xie, and J. Bishop. Code Hunt: Gamifying Teaching and Learning of Computer Science at Scale. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 221–222. ACM, 2014.

[42] N. Truong, P. Bancroft, and P. Roe. Learning to Program Through the Web. *ACM SIGCSE Bulletin*, 37(3):9–13, 2005.

[43] A. Vihavainen, M. Luukkainen, and J. Kurhila. Multi-faceted Support for MOOC in Programming. In *Proceedings of the 13th Annual Conference on Information Technology Education*, pages 171–176. ACM, 2012.

[44] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel. Scaffolding Students' Learning using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, pages 117–122. ACM, 2013.

[45] B. Vogel-Heuser, S. Rehberger, T. Frank, and T. Aicher. Quality Despite Quantity - Teaching Large Heterogenous Classes in C Programming and Fundamentals in Computer Science. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 367–372. IEEE, 2014.

[46] M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Information and Software Technology*, 55(6):1004–1016, 2013.

[47] Q. Wang, W. Li, and T. Xie. Educational Programming Systems for Learning at Scale. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 177–178. ACM, 2014.

[48] C. Willems, J. Jasper, and C. Meinel. Introducing Hands-On Experience to a Massive Open Online Course on openHPI. In *IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE2013)*, pages 307–313. IEEE, 2013.

[49] L. Wu, G. Liang, S. Kui, and Q. Wang. CEclipse: An Online IDE for Programing in the Cloud. In *IEEE World Congress on Services (SERVICES) 2011*, pages 45–52. IEEE, 2011.

[50] D. Xiao and R. C. Miller. A Multiplayer Online Game for Teaching Software Engineering Practices. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 159–160. ACM, 2014.

[51] J. L. Zachary and P. A. Jensen. Exploiting Value-Added Content in an Online Course: Introducing Programming Concepts via HTML and JavaScript. *ACM SIGCSE Bulletin*, 35(1):396–400, 2003.